**CHATBOT PROJECT**

**Name of Intern: Tanupriya Pathak**

**OVERVIEW:**

This code represents the implementation of a rule-based chatbot that simulates a Delhi Metro Rail Corporation (DMRC) information system. The chatbot is designed to answer frequently asked questions (FAQs) related to the DMRC, provide information about metro routes, and calculate fares for journeys.

An overview of the main components and functionalities of the chatbot:

**Importing Libraries:** The code starts by importing necessary Python libraries, including pandas for handling data, requests for making API requests, difflib for finding closest matches, TensorFlow for deep learning, and transformers for using the DistilBERT model.

**Loading Data:** The code reads two Excel files, "newchatbot.xlsx" and "newstation.xlsx," containing FAQs and station information, respectively. The FAQs are loaded into a dictionary called faq\_dict, where each question maps to its corresponding answer. The station names and codes are loaded into station\_dict.

**DistilBERT Model:** The chatbot uses the DistilBERT model, a pre-trained transformer model, for contextual embeddings and similarity calculations. The model is loaded using the Hugging Face transformers library.

**Contextual Embeddings:** The function get\_contextual\_embeddings converts text input into contextual embeddings using the DistilBERT model. These embeddings represent the semantic meaning of the text.

**Similarity Calculation:** The function find\_most\_similar\_question takes an input question and calculates its similarity with each FAQ question using cosine similarity on their embeddings. It returns the most similar question along with its similarity score.

**Closest Match:** The function find\_closest\_match finds the closest matching station name from the input string using the difflib library.

**Route Information Retrieval:** The function route\_info allows the user to input "From" and "To" station names, and it retrieves route information from the DMRC API using a specific URL. The API response is then parsed to extract information like total time, fare, line, line number, towards station, boarding station information, and interchange station information.

**Main Function:** The main function is the entry point of the chatbot. It repeatedly asks the user for input and processes it accordingly. If the user inputs "route," the chatbot will ask for "From" and "To" station names and then call the route\_info function. If the user inputs "exit," the chatbot will terminate. Otherwise, it will search for the most similar FAQ question using find\_most\_similar\_question and provide an answer if the similarity score exceeds a certain threshold.

**Interaction:** The chatbot interacts with the user, asking for input and providing relevant information based on the user's queries.

**CODE:**

import pandas as pd

import requests

import json

import difflib

import tensorflow as tf

from torch.utils.data import TensorDataset

from scipy.spatial.distance import cosine

from transformers import DistilBertTokenizer, TFDistilBertModel

# Load the DistilBERT tokenizer and model

tokenizer = DistilBertTokenizer.from\_pretrained("distilbert-base-uncased")

model = TFDistilBertModel.from\_pretrained("distilbert-base-uncased", output\_hidden\_states=True)

# Load the FAQ questions and answers from the Excel file

df = pd.read\_excel("newchatbot.xlsx")

faq\_dict = dict(zip(df["Question"], df["Answer"]))

# Load the station names and codes from the Excel file

station\_df = pd.read\_excel("newstation.xlsx")

station\_dict = dict(zip(station\_df["NAME"].str.title(), station\_df["CODE"]))

def get\_contextual\_embeddings(text):

    input\_ids = tokenizer.encode(text, add\_special\_tokens=True)

    inputs = tf.constant([input\_ids])

    outputs = model(inputs)

    hidden\_states = outputs.last\_hidden\_state

    contextual\_embeddings = tf.reduce\_mean(hidden\_states, axis=1).numpy()

    return contextual\_embeddings

def find\_most\_similar\_question(input\_question, faq\_dict):

    input\_embeddings = get\_contextual\_embeddings(input\_question)

    max\_similarity = -1

    most\_similar\_question = None

    for question in faq\_dict:

        question\_embeddings = get\_contextual\_embeddings(question)

        similarity = 1 - cosine(input\_embeddings.flatten(), question\_embeddings.flatten())

        if similarity > max\_similarity:

            max\_similarity = similarity

            most\_similar\_question = question

    return most\_similar\_question, max\_similarity

def find\_closest\_match(input\_str, options):

    closest\_match = difflib.get\_close\_matches(input\_str.title(), options, n=1, cutoff=0.8)

    return closest\_match[0] if closest\_match else None

def route\_info():

    from\_stn = input("From: ").title()

    to\_stn = input("To: ").title()

    print("Please wait...")

    from\_stn\_code = station\_dict.get(from\_stn)

    to\_stn\_code = station\_dict.get(to\_stn)

    if not from\_stn\_code:

        print(f"Could not find a matching station for '{from\_stn}'")

        return

    if not to\_stn\_code:

        print(f"Could not find a matching station for '{to\_stn}'")

        return

    url = "http://139.59.31.166:8000/api/v2/en/station\_route/{from\_stn\_code}/{to\_stn\_code}/least-distance/2023-07-12%2010:36:00.000000"

    url = url.format(from\_stn\_code=from\_stn\_code, to\_stn\_code=to\_stn\_code)

    response = requests.get(url)

    if response.status\_code == 200:

        data = json.loads(response.text)

        stations = data['stations']

        total\_time = data['total\_time']

        fare = data['fare']

        line = data['route'][0]['line']

        line\_number = data['route'][0]['line\_no']

        towards\_station = data['route'][0]['towards\_station']

        print(f"No. of stations: {stations}, Total time: {total\_time}, Total Fare: {fare} .")

        input\_boarding\_info\_askuser = input('Do you want boarding station information: (yes/no) ? ')

        if input\_boarding\_info\_askuser.lower() == 'yes':

            output\_boarding\_info = f"Board at station: {data['route'][0]['start']}, Towards Station: {data['route'][0]['towards\_station']}, in Platform : {data['route'][0]['platform\_name']} "

            print(output\_boarding\_info)

        elif input\_boarding\_info\_askuser.lower() == 'no':

            pass

        else:

            print('Did not match your response')

        if len(data['route']) > 1:

            input\_interchange\_info\_askuser = input('Do you want interchange station information: (yes/no) ? ')

            if input\_interchange\_info\_askuser.lower() == 'yes':

                outputStations = [data['route'][i]['start'] for i in range(1, len(data['route']))]

                print(f"No of Interchange stations are: {len(data['route'])-1}, Namely: {outputStations}")

            elif input\_interchange\_info\_askuser.lower() == 'no':

                pass

            else:

                print('Did not match your response')

    else:

        print(f"Error: {response.status\_code}")

def main():

    similarity\_threshold = 0.85

    while True:

        question = input("Hi! I'm DMRC bot. How may I help you? \nIf your query is related to metro route or fare of journey, type 'route'. \nTo exit the chatbot, type 'exit'. \nOtherwise, enter your question: ")

        if question.lower() == 'route':

            route\_info()

            continue

        elif question.lower() == "exit":

            print("Goodbye!")

            break

        print("Please wait...")

        most\_similar\_question, similarity\_score = find\_most\_similar\_question(question, faq\_dict)

        if most\_similar\_question is not None:

            if similarity\_score >= similarity\_threshold:

                answer = faq\_dict[most\_similar\_question]

                print(answer)

            else:

                print("Sorry, I couldn't find a similar question in the dataset.")

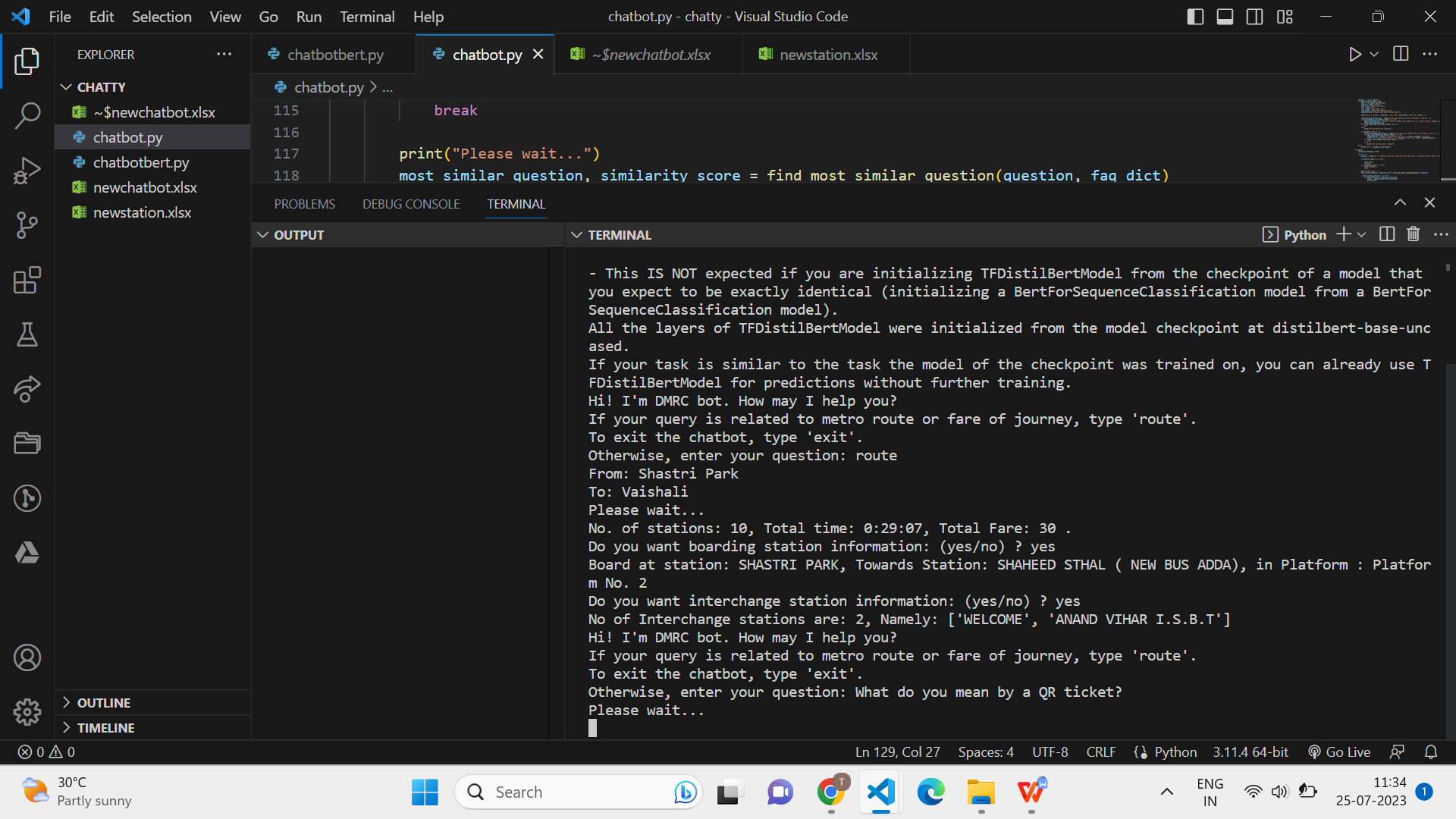
        else:

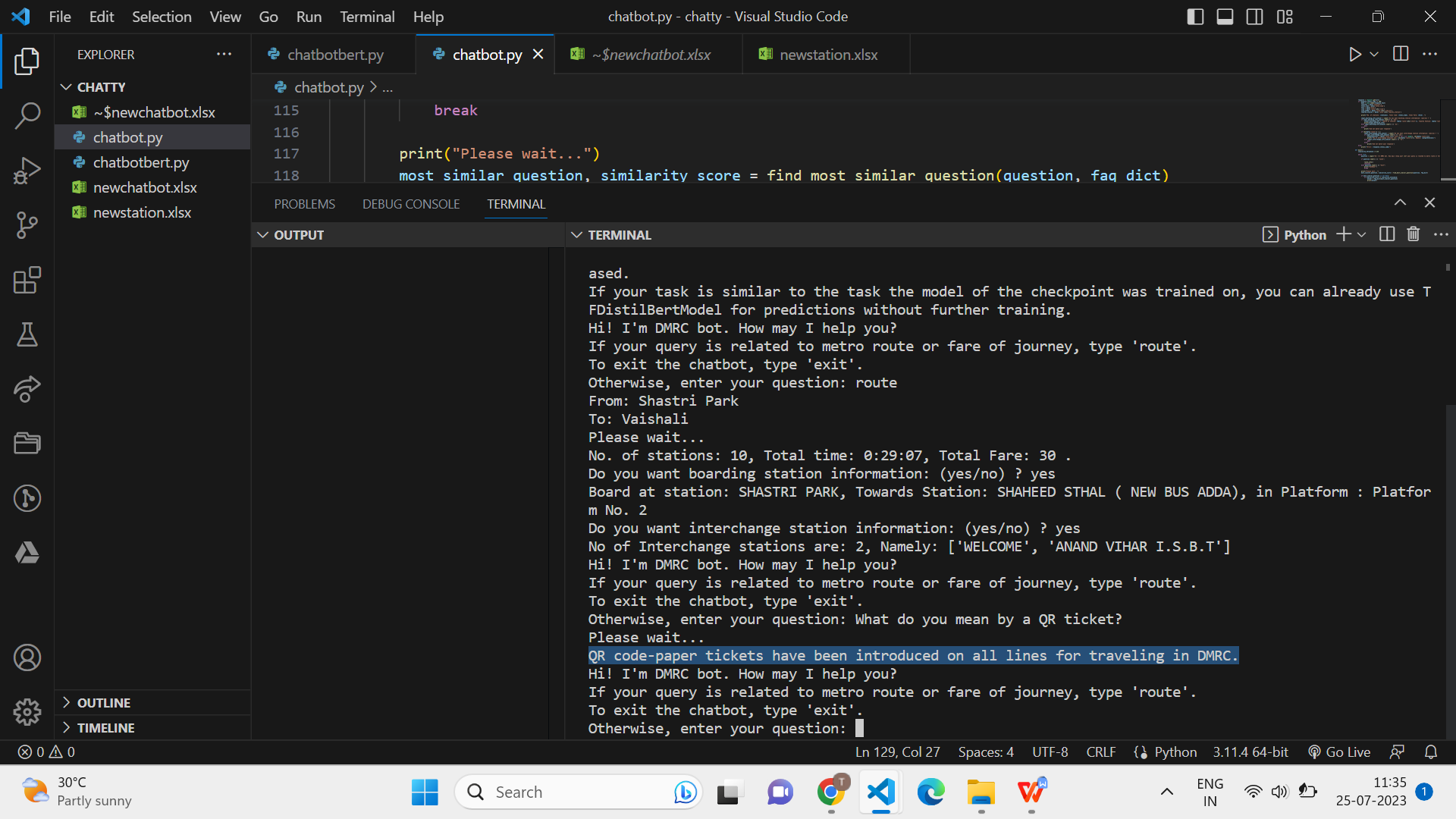
            print("Sorry, I couldn't find a similar question in the dataset.")

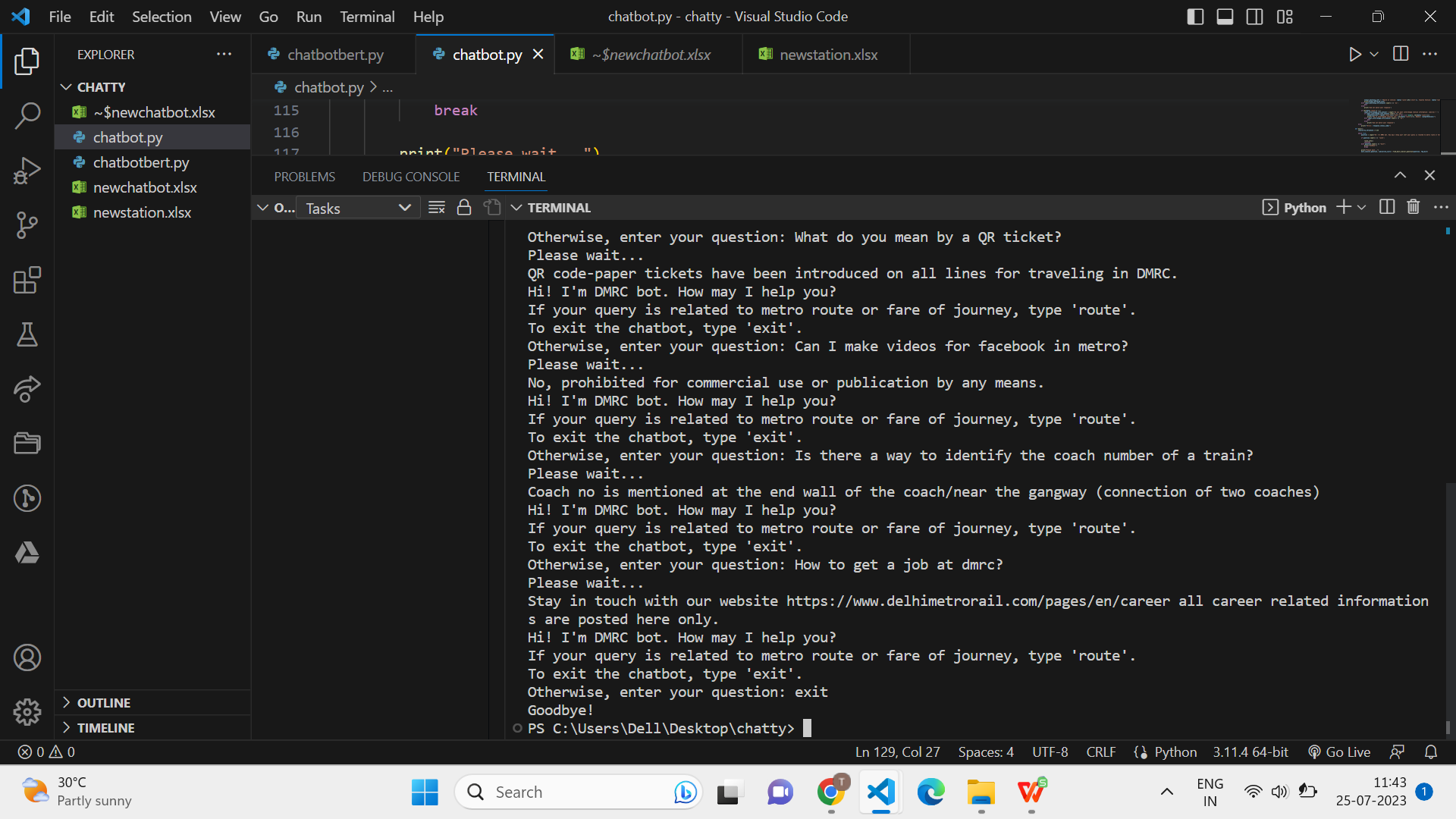
if \_\_name\_\_ == "\_\_main\_\_":

    main()

**SNAPSHOTS:**







**Random Irrelevant Prompts**

